新一代

Visual C++ 2005 程序设计

林俊杰　编著

清华大学出版社

需要全本请访问 www.ertongbook.com

内 容 简 介

本书详细描述了使用Visual C++ 2005与MFC开发Windows窗体应用程序的方法，通过循序渐进的教学模式，一步步教读者构建功能复杂的Win32应用程序。

全书共19章，前3章简单回顾了C++语言的重点并介绍MFC库中与窗口无关的一些基础类，接下来逐章讲述窗口程序设计的基本概念、窗口类CWnd、Windows控件、键盘及鼠标的相关用法、控制菜单的方法、MFC文档/视图结构、绘图及处理位图的方法、打印及控制打印图形尺寸的方法、ActiveX控件。第18、19章是较高级的主题，分别讨论Win32模式下的内存管理、进程及多线程系统，以及registry系统。

本书面向 Visual C++初学者，要求读者必须有 C++语言基础，而且能够操作 Windows，如果在面向对象及数据结构方面有深入的了解则更佳。

版权所有，翻印必究。举报电话：010-62782989 13501256678 13801310933
本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
本书防伪标签采用特殊防伪技术，用户可通过在图案表面涂抹清水，图案消失，水干后图案复现；或将表面膜揭下，放在白纸上用彩笔涂抹，图案在白纸上再现的方法识别真伪。

图书在版编目（CIP）数据

新一代 Visual C++ 2005 程序设计/林俊杰编著. — 北京：
清华大学出版社，2006

ISBN 7-302-13950-4

Ⅰ. 新… Ⅱ. 林… Ⅲ. C 语言－程序设计
Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2006）第 118761 号

出 版 者：清华大学出版社　　**地　　址**：北京清华大学学研大厦
http://www.tup.com.cn　　**邮　　编**：100084
社 总 机：010-62770175　　**客户服务**：010-82896445
组稿编辑：科海
文稿编辑：何武
封面设计：林陶
版式设计：科海
印 刷 者：北京市耀华印刷有限公司
发 行 者：新华书店总店北京发行所
开　　本：787×1092　1/16　　**印张**：30.75　　**字数**：748 千字
版　　次：2006 年 11 月第 1 版　　2006 年 11 月第 1 次印刷
书　　号：ISBN 7-302-13950-4/TP・8386
印　　数：0 001~4 000
定　　价：49.00 元

本书如存在文字不清、漏印以及缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。联系电话：（010）82896445

目　录

第 1 章　综述 1

1.1　本书结构 1
1.2　本书约定 2

第 2 章　C++重点回顾 3

2.1　类、对象与实例 3
2.2　构造函数（Constructor） 3
2.3　匿名实例（Nameless Instance） 4
2.4　虚函数 5
2.5　异常处理 7

第 3 章　基本概念与基础 MFC 类 10

3.1　什么是 Win32 API 10
3.2　控制台模式的应用程序——Win32 Console Mode 10
3.3　基本数据类型 11
3.4　什么是 Unicode 14
3.5　使用 MFC 类库 15
3.5.1　MFC 的根 CObject 15
3.5.2　功能强大的字符串类 CString 15
3.5.3　CFile 家族 17
3.5.4　CException 异常类家族 19
3.5.5　通用的列表类 CList 23

第 4 章　窗口程序的基本概念 25

4.1　文字终端与 GUI 界面 25
4.2　组成要素 26
4.2.1　应用程序 26
4.2.2　窗口 27
4.2.3　消息传递机制 30

第 5 章　创建第一个窗口程序 32

5.1　使用 Visual C++ 32
5.1.1　创建新项目 32
5.1.2　设置项目选项 34
5.1.3　创建新 CPP 文件 35
5.1.4　编译与执行示例程序 36
5.1.5　调试 36
5.2　示例程序 37
5.2.1　HelloApp 38
5.2.2　MyFrameWindow 39
5.3　增加资源文件 41
5.3.1　创建新的资源文件 41
5.3.2　符号与资源的包含文件（Include Files） 41
5.3.3　插入资源 42
5.3.4　编辑位图资源 43
5.3.5　编辑图标资源 44
5.3.6　编辑菜单资源 44
5.3.7　引用资源 46
5.4　用 MFC Application Wizard 产生程序 50
5.4.1　用 MFC Application Wizard 产生项目 51
5.4.2　产生了什么程序代码 54
5.4.3　在 CChildView 内显示文字 56

第 6 章　窗口的产生与处理 58

6.1　窗口类 58
6.2　产生 CWnd 对象 59
6.2.1　CWnd::Create() 60
6.2.2　CWnd::CreateEx() 62
6.2.3　WM_CREATE 消息 63
6.2.4　CREATESTRUCT 63
6.3　处理消息 64
6.3.1　映射表 64
6.3.2　给窗口发送消息 65
6.4　关闭窗口 67
6.5　窗口位置与尺寸 69
6.5.1　获取窗口的位置及大小 69

6.5.2 移动窗口与改变大小 69
6.5.3 移动窗口、改变大小及垂直相对位置 69
6.5.4 移动与改变大小的消息.....71
6.5.5 改变窗口显示的状态 72
6.5.6 工作区的相关消息 72
6.5.7 坐标系统的转换.....73
6.6 窗口的状态与样式.....74
6.6.1 图标化.....74
6.6.2 可见性.....74
6.6.3 放大.....74
6.6.4 使能.....74
6.6.5 活动与非活动.....75
6.6.6 子窗口.....75
6.6.7 窗口的样式 76
6.7 工作区显示.....76
6.8 有滚动条的窗口 78
6.8.1 滚动条.....78
6.8.2 滚动条的范围.....79
6.8.3 滚动块的位置.....80
6.8.4 滚动条的消息.....80
6.8.5 滚动工作区 81
6.9 CWnd 与句柄.....81
6.10 窗口文本.....82

第 7 章 对话框 83

7.1 基础知识.....83
7.1.1 对话框与一般窗口的区别.....83
7.1.2 必备的 OK、Cancel 键.....85
7.1.3 控件.....86
7.1.4 控件的通知消息.....86
7.2 编辑器与列表框的字符串交换.....87
7.2.1 创建项目 87
7.2.2 对话框资源编辑器 88
7.2.3 设计对话框的外观 90
7.2.4 响应各种事件.....92
7.2.5 运行的细节 97
7.3 消息框 100
7.4 多页对话框.....103
7.4.1 多页对话框的原理 103
7.4.2 创建多页对话框的程序.....103
7.4.3 CPropertySheet 的构造函数.....104
7.4.4 CPropertySheet 的成员函数.....104
7.4.5 CPropertyPage 的构造函数.....105
7.4.6 CPropertyPage 的成员函数.....105
7.4.7 CPropertyPage 可以覆盖的成员函数.....105
7.4.8 示例程序 106
7.5 通用对话框.....108
7.5.1 文件对话框 109
7.5.2 颜色对话框 113
7.5.3 其他的通用对话框 114

第 8 章 键盘、鼠标与时间 115

8.1 消息与输入焦点 115
8.1.1 改变输入焦点.....115
8.1.2 与输入焦点有关的消息.....116
8.2 键盘的消息.....116
8.2.1 ASCII 码、虚拟键码与扫描码 117
8.2.2 消息.....117
8.3 鼠标.....119
8.3.1 鼠标消息 119
8.3.2 更换鼠标的光标.....121
8.3.3 显示等待光标.....122
8.3.4 获取鼠标的控制权 123
8.4 特殊的状况.....124
8.4.1 在中文窗口下的文本输入.....124
8.4.2 Windows 的功能键 125
8.5 示例：模拟打字程序.....126
8.5.1 获取消息 126
8.5.2 文本光标 127
8.5.3 鼠标光标 128
8.5.4 显示输入的字符.....129
8.6 计时器消息与时间.....133
8.7 设计鼠标光标.....134
8.7.1 绘制手掌型光标.....134
8.7.2 光标的热点 135
8.7.3 更换鼠标光标.....135

第 9 章　菜单、工具栏与快捷键.....136

9.1　菜单.....136
9.1.1　命令消息.....137
9.1.2　使用菜单.....140
9.1.3　系统菜单.....140
9.1.4　弹出式菜单.....141
9.1.5　CMenu 的成员函数.....142
9.1.6　示例.....147
9.2　快捷键.....151
9.2.1　快捷键的工作方式.....151
9.2.2　编辑快捷键表资源.....152
9.2.3　示例.....153
9.3　工具栏与状态栏.....154
9.3.1　创建工具栏.....155
9.3.2　编辑工具栏资源.....157
9.3.3　创建状态栏.....158
9.3.4　在状态栏显示时间.....159
9.3.5　工具栏与状态栏的开关.....159
9.3.6　状态栏与 Tooltip 中的帮助说明文字.....160

第 10 章　文档、视图与框架窗口.....163

10.1　“文档/视图”模型.....163
10.1.1　文档与视图（Document/View）.....163
10.1.2　单文档界面与多文档界面.....164
10.2　文档、视图与框架窗口间的关系.....166
10.2.1　一个标准的 SDI 示例.....166
10.2.2　编辑字符串表（String Table）.....170
10.3　CDocument 类.....175
10.3.1　View 的管理.....175
10.3.2　创建或打开文档.....176
10.3.3　保存文件.....177
10.3.4　关闭文件.....178
10.3.5　Serialize.....178
10.3.6　应该被重载的函数.....182
10.3.7　其他成员.....183
10.4　CView 类.....183
10.5　CScrollView 类.....184
10.5.1　CScrollView 的工作原理.....184
10.5.2　CScrollView 的成员函数.....184
10.6　SDI 框架窗口 CFrameWnd.....186
10.6.1　重要的成员函数.....186
10.6.2　菜单的自动 enable / disable.....187
10.7　示例：拉线绘图.....187
10.7.1　CList.....187
10.7.2　鼠标的操作.....188
10.7.3　Serialize.....188
10.7.4　更换鼠标光标.....189
10.7.5　其他细节.....189
10.7.6　改用 CScrollView.....197
10.8　MDI.....198
10.8.1　一个标准的 MDI 示例.....198
10.8.2　排列 MDI 子窗口.....203
10.8.3　管理 MDI 子窗口.....203
10.8.4　CMDIFrameWnd 的自动功能.....204
10.8.5　Window 选项.....204
10.8.6　最近使用的文档 MRU.....204
10.8.7　与 MDI 有关的消息.....205
10.9　MDI 化的 LineArt.....206
10.10　多视图类的 MDI.....213
10.11　拆分式窗口(Splitter Window).....221
10.11.1　包含“动态拆分窗口”的 SDI 标准示例.....222
10.11.2　包含“静态拆分窗口”的 SDI 示例.....225
10.12　更多不同的 View 类.....232
10.12.1　另一类对话框 CFormView.....232
10.12.2　可以直接编辑文字的 CEditView.....233

第 11 章　文档视图模式与 MFC Application Wizard.....239

11.1　SDI 项目.....239
11.1.1　使用 MFC Application Wizard 创建项目.....239
11.1.2　新项目中有哪些类.....241

需要全本请访问 www.ertongbook.com

11.1.3 用 ClassWizard 重做 SDI 版的 LineArt.....241
11.2 MDI 项目.....247
11.2.1 使用 MFC Application Wizard 创建 MDI 项目.....247
11.2.2 新项目中有哪些类.....248
第 12 章 控件.....249
12.1 静态类.....249
12.1.1 CStatic::Create().....250
12.1.2 样式.....250
12.1.3 成员函数.....251
12.1.4 CStatic 的通知消息.....252
12.2 按钮.....252
12.2.1 CButton::Create().....252
12.2.2 样式.....252
12.2.3 成员函数.....255
12.2.4 通知消息.....256
12.2.5 示例：计算器.....257
12.3 滚动条 CScrollBar.....268
12.3.1 CScrollBar::Create().....268
12.3.2 样式.....269
12.3.3 成员函数.....269
12.3.4 通知消息.....270
12.3.5 示例：多线程动画.....272
12.4 列表框.....279
12.4.1 样式.....279
12.4.2 成员函数.....281
12.4.3 通知消息.....284
12.5 编辑控件.....284
12.5.1 样式.....284
12.5.2 成员函数.....285
12.5.3 通知消息.....287
12.6 组合框.....287
12.6.1 样式.....288
12.6.2 成员函数.....289
12.6.3 通知消息.....292
12.7 数据交换与校验.....292
12.7.1 简单的示例.....293
12.7.2 常见的 DDX 函数.....296
12.7.3 常见的 DDV 校验函数.....298
12.7.4 在资源编辑器中添加 DDX/DDV 选项.....298
第 13 章 设备上下文与基本绘图工具.....300
13.1 概述.....300
13.2 设备上下文类 CDC.....300
13.2.1 创建与删除 CDC 对象.....300
13.2.2 CDC 的内置资源对象.....302
13.2.3 查询 DC 的相关信息.....303
13.2.4 CDC 的其他成员函数.....305
13.3 CWindowDC 与 CClientDC.....305
13.3.1 CWindowDC 的构造函数.....306
13.3.2 CClientDC 的构造函数.....306
13.4 画笔与画刷.....306
13.4.1 画笔.....306
13.4.2 画刷.....308
13.5 基本绘图函数.....309
13.5.1 点.....309
13.5.2 线.....309
13.5.3 弧线.....310
13.5.4 椭圆.....311
13.5.5 矩形.....311
13.5.6 饼图.....311
13.5.7 多边形.....312
13.5.8 其他.....312
13.6 色彩与调色板.....313
13.6.1 显卡的概念.....313
13.6.2 256 色模式.....313
13.6.3 32K，64K 及 16M 色模式.....314
13.6.4 逻辑调色板和硬件调色板.....314
13.6.5 常用来表示“颜色”的结构与宏.....315
13.7 使用调色板.....317
13.7.1 创建调色板对象.....317
13.7.2 将调色板对象指派给 DC.....317
13.7.3 “实现”（Realize）调色板.....317
13.7.4 画图.....318
13.7.5 响应系统消息.....318

13.7.6　调色板的其他成员函数.....319
13.8　字体与文本输出.....319
13.8.1　基本知识.....319
13.8.2　如何描述“字体”.....320
13.8.3　CFont 类.....323
13.8.4　多样的文本输出方式.....324
13.8.5　文本输出的格式与属性.....328
13.9　示例：时钟.....331
第 14 章　位图.....337
14.1　简介.....337
14.1.1　设备独立位图（DIB）.....337
14.1.2　设备相关位图（DDB）.....338
14.2　CBitmap 类.....339
14.2.1　创建 DDB.....339
14.2.2　CBitmap 成员函数.....339
14.2.3　与 DDB 有关的函数.....340
14.3　设备无关位图类（DIB）.....341
14.3.1　DIB 的结构.....341
14.3.2　BITMAPFILEHEADER.....342
14.3.3　BITMAPINFO.....342
14.3.4　BITMAPINFOHEADER.....342
14.3.5　RGBQUAD.....343
14.3.6　BMP 文件的格式.....344
14.3.7　Win32 API 与 DIB 有关的函数.....344
14.3.8　DIB 与 DDB 的转换.....345
14.4　处理 DIB 的示例.....345
14.4.1　自己设计 CDib 类.....346
14.4.2　主程序.....357
第 15 章　打印与坐标系.....369
15.1　打印.....369
15.1.1　设置打印机的状态.....369
15.1.2　通过 CView 打印.....370
15.1.3　示例.....374
15.2　比例、原点与方向.....376
15.2.1　改变映射模式.....377
15.2.2　理想与现实.....377
15.2.3　示例.....378
15.3　更大的灵活性.....381
15.4　打印预览.....383
第 16 章　通用控件.....384
16.1　概述.....384
16.1.1　通用控件的种类.....384
16.1.2　使用通用控件前的初始化.....386
16.1.3　通用控件的共性.....386
16.1.4　通用控件的通知消息.....387
16.1.5　通用控件都会产生的通知消息.....388
16.2　CSliderCtrl.....388
16.2.1　CSliderCtrl 的样式.....388
16.2.2　CSliderCtrl 成员函数.....389
16.2.3　CSliderCtrl 的通知消息.....390
16.3　CSpinButtonCtrl.....391
16.3.1　CSpinButtonCtrl 的样式.....391
16.3.2　CSpinButtonCtrl 的用法.....391
16.3.3　CSpinButtonCtrl 的其他成员函数.....392
16.3.4　CSpinButtonCtrl 的通知消息.....392
16.4　CHeaderCtrl.....392
16.4.1　CHeaderCtrl 的样式.....392
16.4.2　使用 CHeaderCtrl.....393
16.4.3　其他的成员函数.....394
16.4.4　CHeaderCtrl 的通知消息.....395
16.5　CAnimateCtrl.....395
16.5.1　CAnimateCtrl 的样式.....395
16.5.2　CAnimateCtrl 的成员函数.....395
16.5.3　CAnimateCtrl 的通知消息.....396
16.5.4　示例.....396
16.6　CProgressCtrl.....398
16.6.1　CProgressCtrl 的用法.....398
16.6.2　示例.....398
16.7　CTreeCtrl、CTreeView 与 CImageList.....398
16.7.1　使用 CImageList.....399
16.7.2　CTreeCtrl 的样式.....400
16.7.3　CTreeCtrl 的用法.....400
16.7.4　其他的成员函数.....402

16.7.5 CTreeCtrl 的通知消息.....405
16.7.6 CTreeView.....406
16.7.7 示例：显示磁盘的树状目录.....406
16.8 CListCtrl 与 CListView.....416
16.8.1 CListCtrl 的样式.....417
16.8.2 CListCtrl 的用法.....418
16.8.3 CListCtrl 的成员函数.....421
16.8.4 CListCtrl 的通知消息.....424
16.8.5 CListView.....425
16.8.6 示例：显示目录中的文件列表.....426
16.9 示例：程序管理器.....435
16.10 CIPAddressCtrl.....438
16.10.1 成员函数.....438
16.10.2 通知消息.....439
第 17 章 使用 ActiveX 控件.....440
17.1 ActiveX 控件的由来.....440
17.2 使用 ActiveX 控件.....441
17.2.1 安装 ActiveX 控件.....442
17.2.2 MCI32.OCX 简介.....443
17.3 设计 ActiveX 容器.....443
17.3.1 启动 ActiveX.....443
17.3.2 插入 ActiveX 控件.....444
17.3.3 在对话框中使用 MCI 控件.....445
17.3.4 在程序中控制 MCI 控件.....446
第 18 章 进程、线程与内存.....448
18.1 Win32 的内存管理.....448
18.2 Process 简介.....452
18.2.1 产生子进程.....453
18.2.2 与 process 有关的信息.....456
18.2.3 结束 process.....457
18.3 Thread 简介.....457
18.3.1 产生 Child Thread.....458
18.3.2 Thread 的信息.....458
18.3.3 终止 Thread.....458
18.4 同步（Synchronization）.....458
18.4.1 Critical Section.....460
18.4.2 等待对象的状态改变.....463
18.4.3 Mutex.....464
18.4.4 Event.....466
18.5 进程间通信（IPC）.....469
18.6 CWinThread.....472
18.6.1 GUI Thread 与非 GUI Thread.....472
18.6.2 利用 CWinThread 产生 Thread 的流程.....472
18.6.3 其他的 CWinThread 成员.....473
第 19 章 Profile 与 Registry.....474
19.1 什么是 Profile.....474
19.2 什么是 Registry.....475
19.2.1 创建新的 key.....476
19.2.2 打开已存在的 entry.....477
19.2.3 写入数据.....477
19.2.4 读取数据.....478
19.2.5 删除 key 或 entry.....478
19.2.6 关闭 Key.....479

第1章

综　　述

初学者经常提出这样的问题：是学 C++好还是学 Visual C++好？

其实，这是个很奇怪的问法，就好像在问：是学开车好还是学开福特车好？开车是一种普遍的技术，用这个技术可以控制福特车或奥迪车，这在技术上没有很大的差别，最多只是一些开关的位置不同而已。同样的道理，C++是一种程序语言，现在已经成为 ISO 标准，是一种普遍的技术。Visual C++只是一种协助用户使用 C++语言设计软件的工具。

类似的工具还有很多，例如 Borland C++ Builder，甚至还有免费的 GNU C++，它们都是协助用户使用 C++设计软件的工具。既然可以使用免费的 GNU C++，为什么还要花钱买昂贵的 Visual C++呢？

首先，Visual C++有功能强大的集成开发环境，在此环境中可以方便地进行项目管理、编写 C++源程序、编译、调试，而这些在 GNU C++中都不具备。更重要的是，Visual C++提供了一套称为 Microsoft Foundation Class（MFC）的类库（classes library），这套由微软开发的库已经成为设计窗口应用程序的准工业标准了。

标准 C 语言库提供了 400 种库函数，例如 printf()、fopen()等，这些函数可以实现数据存取、数学计算、字符串处理等功能。MFC 包含了上百种事先用 C++写好的类，将这些类组合起来，可以编写功能强大的 Windows 应用程序。

除了 MFC 之外，其他厂商也有类似的产品，例如 Borland C++ Builder 的 VCL，也是用 C++设计的库，可以协助用户快速地开发窗口应用程序。这些库各有所长，MFC 的最大优势在于能抢先支持各种新功能，如 ActiveX 等，但是在面向对象技术方面略逊一筹，例如，MFC 的类难以实现多重继承。

1.1　本书结构

本书的主题是讲述如何用 MFC 设计窗口应用程序。在阅读本书之前，要求读者必须学会 C++语言，而且能够操作 Windows，如果在面向对象及数据结构方面有深入的了解则更好。

本书第 2 章简单地回顾了一些 C++语言的重点；第 3 章介绍 MFC 库中与窗口无关的一些基础类；第 4 章讲述窗口程序设计的基本概念。

第 5 章将实际动手设计一个最简单的窗口应用程序。接下来，第 6 章详细介绍窗口类 CWnd，不过第 6 章的后半部分内容稍微有点深，初学者可以先跳过不看。第 7 章讲解用 Visual C++设计以对话框为主的应用程序，并且在此处介绍 Windows 的控件。第 8 章讲述输入设备——键盘及鼠标的相关用法。第 9 章介绍控制菜单的方法。第 10 章是本书最重要的一章，介绍 MFC 的文档/视图结构，请务必搞清楚。第 11 章介绍用 MFC Application Wizard 辅助设计文档/视图结构软件的方法。第 12 章是第 7 章的延伸，详细地介绍了各种控件的用法，初学者可先跳过不看。

第 13 章的主题是绘图，Windows 绘图的所有知识都可以在这里找到。第 14 章介绍处理位图的方法。在该章后半部分，笔者设计了一个可以处理 Windows 常见的*.BMP 文档的类，十分有用，请务必仔细阅读。仅在屏幕上显示图形还不够，有时需要将其打印出来，第 15 章介绍打印以及控制打印图形尺寸的方法。

第 16 章介绍在 Windows 95 之后才出现的通用控件，该章的范例就是逐步创建一个类似文件管理器的应用程序。第 17 章介绍如何在程序中使用流行的 ActiveX 控件。第 18、19 章是比较高级的主题，分别讨论 Win32 模式下的内存管理、进程及多线程系统，以及 registry 系统，初学者可跳过不看。

1.2 本书约定

File|Open → *如果是指 File 菜单中的 Open 选项，统一以竖线分隔这些选项。*

```
1.  void main()
2.  {
3.  }
```

→ *程序代码左侧的数字是行号。*

int getch(); → *函数的原型(prototype)，也就是函数的名称、返回值类型、参数。*

第2章

C++重点回顾

如果用户对 C++程序语言还不熟悉，那么本章内容刚好适合你。因为笔者的前几本书中都假设读者对 C++有相当程度的了解，不过后来发现写信来问 C++问题的人远比对内容有疑问的人多得多。因此，本书特意增加了这一章，介绍一些 C++语言的重点概念。不过，也不要期待仅靠这一章就可以学会 C++，读者还需要阅读其他介绍 C++的书。

2.1 类、对象与实例

这 3 个名词经常困扰许多人，因此需要对它们进行重点说明。“类”就是种类、品种，比如说狗，就是泛指长了一条尾巴，会吠叫会咬人的哺乳动物，不过每一条狗都是独立、有个性的小生命，有的狗叫小花，有的叫小黄，有的叫来福。因此，狗是一种动物类，而小花、小黄、来福是狗的“实例”；在大多数情况中，实例指的就是“对象”。

2.2 构造函数（Constructor）

Constructor 在这本书中统一称为构造函数。创建实例时，计算机就会自动跳到该实例的构造函数去执行。通常，构造函数中的内容都是和初始化实例相关的程序代码。

不过，构造函数也会有一些奇怪的用法。首先，类可能会有数个父类，而父类也有它自己的构造函数，因此子类必须调用父类的构造函数，以便对从父类继承的部分进行初始化。例如下面这段程序：

```
1.  class Parent
2.  {
3.  public:
4.       int parent_var;
5.
6.       Parent ()          { parent_var = 10; }
7.       Parent ( int init ){ parent_var = init; }
8.  };
9.
10. class Child : public Parent
11. {
12. public:
```

需要全本请访问 www.ertongbook.com

```
13.      int child_var;
14.
15.      Child ()                        { child_var = 23; }
16.      Child ( int i ) : Parent ( i ) { child_var = 24; }
17.      Child ( int i, int j )
18.          : Parent ( i ), child_var ( j )    {}
19. };
20.
21. Child inst_1;
22. Child inst_2 ( 100 );
23. Child inst_3 ( 100, 200 );
```

在这段程序中，如果产生 Child 的实例 inst_1，inst_1 的成员变量 parent_var 会被设成 10，child_var 会被设成 23，这是因为计算机会跳到构造函数 Child::Child()中执行，而在将 child_var 设成 23 之前，会先跳到 Parent::Parent()中执行。inst_2 的 parent_var 会被设成 100，因为 Child::Child(int)冒号后的指令要求计算机先运行 Parent::Parent(int)，再把 child_var 设置成 24。

Child 第 3 个构造函数就更奇怪了，在冒号之后调用 Parent 构造函数还不难了解，但是 child_var(j)就有点奇怪。原来这一行的作用是调用 child_var 对象的构造函数，因为它只是基类，因此等价于 child_var = j。

2.3 匿名实例（Nameless Instance）

请看下面这段程序：

```
1.  int GetValue ( Child& object )
2.  {
3.       return object.child_var;
4.  }
5.
6.  void Trial ()
7.  {
8.       int x;
9.       x = GetValue ( Child ( 4, 5 ) );
10. }
```

在这段程序中，函数 GetValue(Child&)的功能是返回对象 object 成员变量 child_var 的值。那么，函数 Trial()的变量 x 会是多少呢？

最奇怪的是第 9 行，GetValue()的参数竟然只有 Child(4,5)，这是怎么回事？其实这种用法等价于：

```
1.  void Trial ()
2.  {
3.       int x;
4.       Child obj ( 4, 5 );
5.       x = GetValue ( obj );
6.  }
```

这段代码首先产生对象 obj，然后将它当成参数传给 GetValue()。前后两段代码的最大

差别在于，前者产生的对象没有名字，而后者产生的对象的名字叫 obj。再看下面一段示例。

```
return HelloApp().Run()
```

它等价于：

```
HelloApp app;
return app.Run();
```

为什么会这样呢？这要从 C++创建对象的方法谈起。在第 2 段代码中，执行到 HelloApp app 这一行时，计算机会在内存中分配一块空间当做 app 的主体。在我们的概念中，变量 app 就代表这块内存。反过来看第一段程序代码，从左向右看，当计算机执行到 HelloApp() 这一行时，它也会在内存中分配一块空间当做 HelloApp 实例的主体，只是没有一个类似 app 的变量来代表这块内存中的对象。即使如此，这个对象还是存在的，只不过它没有名字。访问此对象的唯一方法就是紧接在后面加一个点，再加上成员的名称。因此，说构造函数没有返回值是不正确的，其实构造函数的返回值就是对象本身。

2.4　虚函数

如果在类的成员函数之前加了 virtual 修饰符，那么该函数就称为虚函数（virtual function）。例如在下面这段程序中，Object::foo()就是虚函数：

```
class Object
{
    virtual void foo ( void );
};
```

虚函数和非虚函数有什么区别呢？举个真实世界中的例子，儿子的姓氏和手都是从父亲继承来的，虽然同样是继承来的东西，但儿子的姓氏实际上还是沿用父亲的姓氏，不过手却是儿子自己的手。在这个例子中，姓氏不是虚函数，只有手才是虚函数。

真实世界的例子或许不足以恰当地说明 C++的世界。再看一段示例程序：

```
1.  class Father
2.  {
3.      protected:
4.      int     my_firstname()      { return 100; }
5.      virtual int my_hand()       { return 101; }
6.
7.      public:
8.      int         get_firstname()     { return my_firstname(); }
9.      int         get_hand()          { return my_hand(); }
10. };
11.
12. class Kid : public Father
13. {
14.     protected:
15.
16.     int     my_firstname()      { return 102; }
17.     virtual int my_hand()       { return 103; }
18. };
```

```
19.
20. void Test ()
21. {
22.     int val1, val2, va3;
23.     Kid kid;
24.     Father *ptr;
25.
26.     var1 = kid.get_firstname();
27.     var2 = kid.get_hand();
28.     ptr = new Kid;
29.     var3 = ptr->get_hand();
30.     delete ptr;
31. }
```

类 Father 有两个 protected 成员函数：my_firstname()与 my_hand()。它的 public 成员函数 get_firstname()是调用 my_firstname()后再返回其值，get_hand()是调用 my_hand()后再返回其值。继承自类 Father 的新类 Kid，理所当然地继承了 Father 所有的成员函数，不过在 Kid 中又重新定义了 my_firstname()与 my_hand()这两个函数，这种行为称为重载（override）。

在函数 Test()中有个 Kid 的实例 kid。在第 28 行调用了 kid.get_firstname()，因为这个函数继承自 Father，所以它理当会调用成员函数 my_firstname()。但实际上它到底会调用谁的 my_firstname()？是 Father::my_firstname()还是 Kid::my_firstname()？

再看第 29 行，它调用 kid.get_hand()。毫无疑问，get_hand()还会再调用 my_hand()。同样的问题又发生了，它到底是调用 Father::my_hand()还是 Kid::my_hand()？

应该已经注意到 my_hand()是虚函数，而 my_firstname()不是，因此 get_hand()、get_firstname()两个函数的行为必然是不同的。因为 my_hand()是虚函数，所以 get_hand()会调用它所属的那个对象的 my_hand()。在 Father 中，get_hand()调用的是 Father::my_hand()，而在 Kid 中，get_hand()是调用 Kid::my_hand()。get_hand()选择 my_hand()的方式称为动态确定。

另一方面，因为 my_firstname()不是虚函数，因此 get_firstname()只会调用 Father::my_firstname()，而不会调用 Kid::my_firstname()。get_firstname() 选择 my_firstname()的方式称为静态确定。

由于调用虚函数时，计算机会动态确定要调用哪个函数，因此这种调用方式又被称为动态绑定（dynamic binding），而调用非虚函数时的行为则被称为静态绑定（static binding）。

觉得神奇吗？其实很简单，编译器会为每个类创建一个称为虚函数表（virtual function table）的表格，这个表格由函数指针组成，每个指针都指向一个虚函数，因此调用虚函数时，计算机会先检查这个表，找出函数指针后再跳到指定地址去执行，这样就形成类似的动态效果。严格地说，这还不够动态，Java 才是真正的动态绑定。

虚函数在本书、MFC 以及其他的类库中占有十分重要的地位。这些类库在设计时并不知道程序员会如何运用它们，因此总得留一些进入点以便插入、更改或者扩展程序代码，在以后的章节代码中，需要修改的地方通常都是虚函数，只要在从库派生的新类中重载虚函数，加上新功能，就可以构造出所需的应用程序。

2.5　异常处理

在传统的 C/C++程序设计中，函数的返回值扮演了多重角色，它有时代表了计算的结果（例如 sin()、cos()这类函数的返回值就是三角函数值），有时又代表出现的错误（例如 ferror()在文件访问发生错误时会返回非零值），甚至还有更复杂的情况，返回值综合了以上两种用途。不同公司的产品、不同时代的类库，就有不同的设计哲学，这给程序员进行程序设计增加了难度。

顺序（例如一般的算术指令）、循环（例如 for）和分支（例如 if...then...else）是结构化程序的 3 大要素，也就是说，一个遵循结构化规则的程序，不会用到 goto 语句，并且，所有用到的指令都不出以上 3 大类。

也许有人会问：为什么不能用 goto？大多数人认为源程序是写给计算机的一串指令，其实不然！源程序是人们思考后有规则、有系统地写下的剧本，这个剧本经过编译器翻译成机器码后才交给计算机执行。因此，源程序其实是写给人看的，而不是计算机，因此，如何让源程序容易被人看懂，最好连没受过训练的 3 岁小孩也能看懂，这是许多人的梦想，所以我们才会不断强调结构化程序的重要性，这也是不使用 goto 的原因。

有些人认为仅靠顺序、循环和分支 3 类指令还不足以写出明晰易懂的程序，如果可以把错误处理的部分独立于正常流程之外，源程序的结构性将会更明确，因此新一代的程序语言开始添加错误处理指令，其中包括新版的 C++及 Java 等。

```
try
{
     ......
     throw ObjectX;
     ......
}
catch ( Class1 Object )
{
     .....
}
catch ( Class2 Object )
{
     .....
}
```

C++异常处理的基本形式与前面的程序代码差不多，也需要用到 try、catch、throw 这 3 个指令。在 try 之后的一对大括号包含了一段程序代码（其中可以再包含一层 try...catch...），在这一段程序代码中如果发生了异常，计算机就会自动跳到 try 大括号后面的 catch 代码块中执行。

如何产生异常呢？只要用 throw 指令抛出一个对象即可。异常对象被抛出之后，计算机会自动找到最近一层的 catch()处理这个异常对象。每个 catch 只能接受某类异常对象，例如，在上面的程序代码中，第一个 catch 只能接受类为 Class1 的对象，第二个 catch 只能接受类为 Class2 的对象，如果被抛出的对象和某个 catch 所能接受的类相同，或者属于它的

子类，计算机就自动跳到这个 catch 后的指令开始执行，执行完毕就退出此 catch 所属的 try 区域，从 try 块下第一条指令开始执行。

```
1.  #include <stdio.h>
2.
3.  class MyError
4.  {
5.  public:
6.       int ch;
7.       MyError ( int c ) { ch = c; }
8.  };
9.
10. void ErrorFunc()
11. {
12.      throw int ( 5 );
13. }
14.
15. void main()
16. {
17.      int ch;
18.      try
19.      {
20.          ch = getch();
21.          if ( ch == 'a' ) throw MyError(ch);
22.          ErrorFunc();
23.      }
24.      catch ( int intErr )
25.      {
26.          puts ( "Error object: int" );
27.      }
28.      catch ( MyError myErr )
29.      {
30.          puts ( "Error objec: MyError" );
31.      }
32. }
```

在上面这段示例中，在 try 区域中先调用 getch()，从键盘读取一个字符。如果用户键入“a”，就抛出一个动态产生的 MyError 实例，这时，计算机会跳到第 28 行执行，因为这个 catch 可接受 MyError 类的对象。如果不是“a”，就执行第 22 行的 ErrorFunc()函数，这个函数惟一的指令就是抛出一个整数对象①，因为 ErrorFunc()本身没有 catch 区域可以处理这个整数对象，所以就继续往上一层查找，即跳回调用它的函数 main()，找到了第一个接受 int 异常对象的 catch，因此会跳到第 26 行执行。

这种结构的最大好处就是，可以把处理错误的程序代码与正常的程序流程分隔开，正常的程序写在 try 区域中，一旦产生错误，就离开 try 跳到相邻的 catch 中执行，因此两个部分泾渭分明。另一方面，try...catch...结构打破了函数的限制，例如，在 ErrorFunc()中发生的错误就可以交给调用它的 main()来处理，每个函数不需要准备自己的错误处理程序，也不必担心如何把错误传递给适当的处理函数，它只要抛出异常对象即可。

① 虽然 int 是个基本数据类型，但也可以看作是一个对象。

但是，如果抛出某个异常对象，却没有恰当的 catch 处理它该怎么办？Visual C++和其他大多数的 C++工具的处理方法就是显示一个错误消息，然后停止该程序的执行。

那么，有没有可以接收各种异常的通用处理过程呢？有，只要用 3 个句点“...”取代 catch 中的声明就可以了，例如下面的这段程序代码：

```
1.  try
2.  {
3.  .....
4.  }
5.  catch (...)
6.  {
7.  ...
8.  }
```

这样，只要 try 区域中发生了无法处理的错误，程序就会跳到第 6~8 行之间的异常处理过程执行。

需要全本请访问 www.ertongbook.com